HP Encina/9000: Middleware for Constructing Transaction Processing Applications

A transaction processing monitor for distributed transaction processing applications maintains the ACID (atomicity, consistency, isolation, and durability) properties of the transactions and provides recovery facilities for aborting transactions and recovering from system or network failures.

by Pankaj Gupta

Transaction processing systems are widely used by enterprises to support mission-critical applications, such as airline reservation systems and banking applications. These applications need to store and update data reliably, provide concurrent access to the data by hundreds or thousands of users, and maintain the reliability of the data in the presence of failures.

The HP Encina/9000 transaction processing monitor [1] provides the middleware for running transaction processing applications. It maintains ACID (atomicity, consistency, isolation, and durability) properties for transactions (see the glossary on page 65). It ensures that applications that run concurrently will maintain data consistency. Encina/9000 also provides recovery facilities for aborting transactions and recovering from failures such as machine or network crashes.

DCE and Distribution

Encina/9000 provides the ability to write distributed applications. Encina/9000 applications can be written as client/server applications with the client and server possibly running on different machines. Encina/9000 servers can communicate and cooperate with each other in updating data on several different machines.

Distributed applications provide several advantages. The data maintained by an enterprise may itself be distributed because of historical and geographical considerations. Furthermore, distributed applications are able to exploit parallelism by running concurrently on several machines.

Distributed computing offers the advantage of improved performance, availability, and access to distributed data. Performance is improved by spreading the computing among various machines. For example, the application's user interface can be run on a PC while the user code could be split to run on several machines. The use of multiprocessing machines to provide parallelism for multiple users can improve the throughput of the system. Availability can be increased by a distributed system in which replication is used to keep several copies of the data. Access to distributed data or to data that is maintained in several databases is also facilitated by distributed computing.

Data may be distributed because the database becomes too large or the CPU on the database machine becomes a bottleneck. Data can also be distributed to increase availability and improve the response time by keeping the data close to the users accessing it. Finally, data can be distributed to keep separate administrative domains, such as different divisions in a corporation that want to keep their data local.

Encina/9000 uses the Open Software Foundation's DCE [2] (Distributed Computing Environment) as the underlying mechanism for providing distribution. It uses the DCE RPC mechanism to provide client/server communication. Encina/9000 is also very closely tied to DCE naming and security services (see the articles on pages 28 and 41 for more about these services). For example, an Encina/9000 server can be protected from unauthorized use by defining access control lists (ACLs). ACLs contain an encoding of the authorization policy for different users and are enforced by DCE at run time. ACLs are described on page 49. Encina/9000 also makes use of the threading package provided by DCE.

To achieve optimum price and performance, careful consideration needs to be given to how the data and the application are partitioned. Throughput and response times are often the key criteria by which users judge the performance of a system. Encina/9000 provides the flexibility of being able to specify the distribution topology of the application. In addition, users can specify data replication if it will help to ensure higher availability of mission-critical data.

Two-Tiered versus Three-Tiered Architectures

In the past, transaction processing applications were implemented using a two-tiered architecture (see Fig. 1). In this

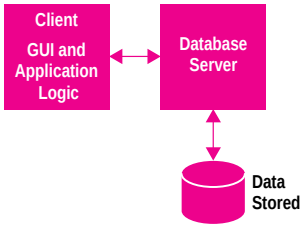


Fig. 1. Two-tiered architecture for transaction processing.

paradigm an application is written as a client, which accesses a database server. The client implements the graphical user interface (GUI) and the application logic. The database server handles access to the data stored in a database.

The advantage of this approach is simplicity. The disadvantage is that it is not scalable beyond a certain point. It is also less flexible and harder to modify to meet new business needs.

Encina/9000 allows the development of applications using a three-tiered architecture like the one shown in Fig. 2. In this paradigm, an application is partitioned into a client that implements the graphical user interface to the user, an application server that implements the business logic of the application, and a database server that implements the database access.

The Encina/9000 three-tiered architecture provides the following advantages over traditional two-tiered architectures:

- Decoupling the GUI from the business logic
- Scalability of the architecture to support a very large number of users and a high transaction throughput
- Accessibility to multiple database servers from an application server
- Freedom from being tied into any particular database vendor
- Tight integration with the distributed computing facilities offered by DCE
- Choice of transactional applications that support any combination of RPC, CPI-C (Common Programming Interface for Communications), and queued message communication
- Ability to retain data on a mainframe or other legacy computer and reengineer by adding HP-UX* application servers, providing lower cost and higher price/performance relative to some mainframe systems.

Three-tiered architectures are more complicated in general but provide greater flexibility of application design and development. Among the reasons why users are willing to give up the simplicity of the two-tiered architecture is the faster response times and the more effective user interfaces provided by the three-tiered architecture. The ability to provide a front-end workstation that supports graphical user interfaces gives an application a more effective user interface. For applications that require access to data distributed across large geographic regions, a three-tiered architecture offers more flexibility to tune the communications to compensate for WAN delays and improve availability. This results in a faster response time because the user is accessing local data most of the time. Propagation of the data to other machines can be queued and performed offline. Therefore, geographically distributed data can be maintained without having to perform expensive distributed two-phase commit protocols online.

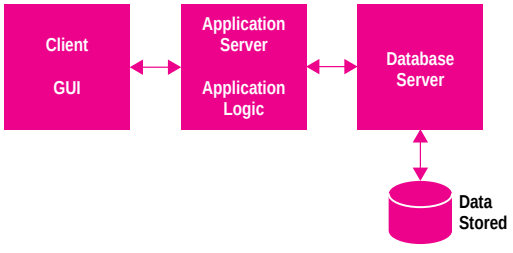


Fig. 2. Three-tiered architecture for transaction processing.

Two-phase commits that happen over wide area networks are expensive and care must be taken when designing distributed applications to minimize the amount of two-phase commits over the network. Queued communications also improve availability. See page 65 for a definition of commit.

Components of Encina/9000

Fig. 3 shows the architecture for the implementation of Encina/9000 that runs on the HP-UX* operating system.

Each of the components shown in Fig. 3 is packaged independently. A machine that runs Encina/9000 clients only can be configured without the Encina/9000 server software. Machines that run Encina/9000 servers must be configured with both the Encina/9000 client and server components.

Encina/9000 applications that can be configured to run on top of the Encina/9000 server component include:

- Peer-to-peer communication, which provides transactional access to data stored on mainframes and workstations running the HP-UX operating system
- Structured file system, which is a record-oriented file system base on the X/Open® ISAM (index sequential access method) standard
- Recoverable queueing service, which provides applications with the ability to enqueue and dequeue data
- Monitor, which is an infrastructure for application development, run-time support, and administration.

The DCE components used by Encina/9000 include: RPC, the directory service, the security service, and threads.

Encina/9000 Toolkit

The Encina/9000 client component is also called the Encina/9000 toolkit executive and the Encina/9000 server component is also called the Encina/9000 toolkit server core. Together these components are called the Encina/9000 toolkit.

Fig. 4 shows the components that make up and support the Encina/9000 toolkit.

Base Development Environment. The lowest layer of the Encina/9000 toolkit is the base development environment. It provides developers of other Encina/9000 components with a uniform set of features independent of the underlying operating system. The base development environment library

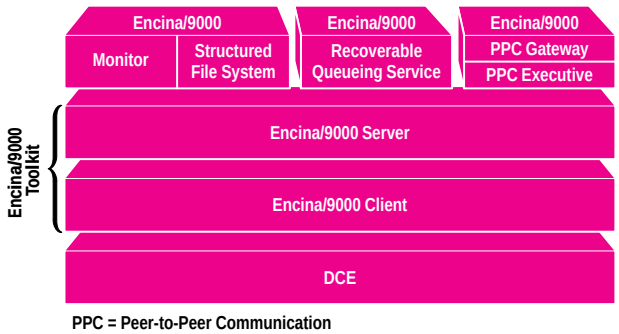


Fig. 3. The architecture of Encina/9000 on the HP-UX operating system.

provides a common platform independent threading interface and an abstraction for low-level functions so that the upper layers that use the base development environment can be independent of differences in the operating system or the hardware platform on which Encina/9000 runs.

The base development environment provides support for multiple threads of execution by using DCE threads. Also, it provides thread-safe routines for the following functionality:

- Memory management
- File I/O
- Process management
- Signal handling
- Timers and alarms
- Native language support.

The base development environment is intended primarily for the development of other Encina/9000 components.

Transaction Manager. The transaction manager provides the ability to demarcate transactions, which means that it is able to specify the beginning, the commit, and the abort of a transaction. Internally it supports a distributed two-phase commit management protocol, including the ability to perform coordinator migration.

The transaction manager supports nested transactions capability,[3] which allows nested transactions to be defined within a top-level transaction. Nested transactions have isolation and durability properties similar to regular transactions, but the abort of a nested transaction does not cause the top-level transaction to abort. This allows a finer granularity of failure isolation in which the main transaction can handle the failure of certain components implemented with a nested transaction. Nested transactions are defined in the glossary on page 65.

The application must be carefully designed since failures such as crashed server nodes, which cause a nested transaction to fail, could in some cases also cause the top-level transaction to fail. The Encina/9000 structured file system provides support for nested transactions for data stored in structured files. However, database vendors like Oracle do not currently support nested transactions in their products, making it impossible to exploit the advantages of Encina/9000's nested transaction capabilities for data stored by these relational databases.

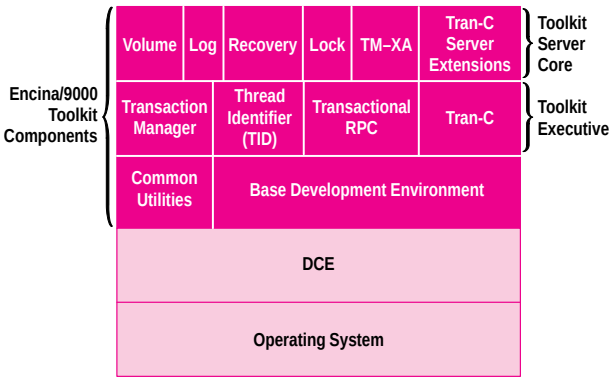


Fig. 4. A detailed view of the components that make up and support the Encina/9000 toolkit.

The Encina/9000 transaction manager provides an application program with the ability to issue callbacks on events related to the transaction's commit protocol. This enables the programmer to write routines that are invoked before the transaction prepares or aborts or after the coordinator decides to abort or commit the transaction.

The transaction manager allows transactions to be heuristically committed by a system administrator. This should only be used in rare cases in which the transaction coordinator is unavailable and the administrator does not want to block access to locked data and has to trade off data availability to avoid possible data inconsistency.

Thread-to-TID. Since Encina/9000 makes use of DCE threads, the work done on behalf of a user transaction can be composed of several different threads. The thread-to-TID service associates a transaction with a thread and maintains the mapping between a thread and a transaction identifier (TID). This service is used by other Encina/9000 components and is rarely used by programmers directly.

Transactional RPC. The Encina/9000 transactional RPC service enhances the DCE RPC mechanism to provide transactional semantics for remote procedure calls. Unlike remote procedure calls, transactional RPCs have once-only semantics. If a transaction performing an RPC commits, then the RPC is guaranteed to have executed once and once only. If the transaction performing an RPC aborts then the RPC is guaranteed not to have executed (if the RPC was executed its effects are undone by the transaction abort).

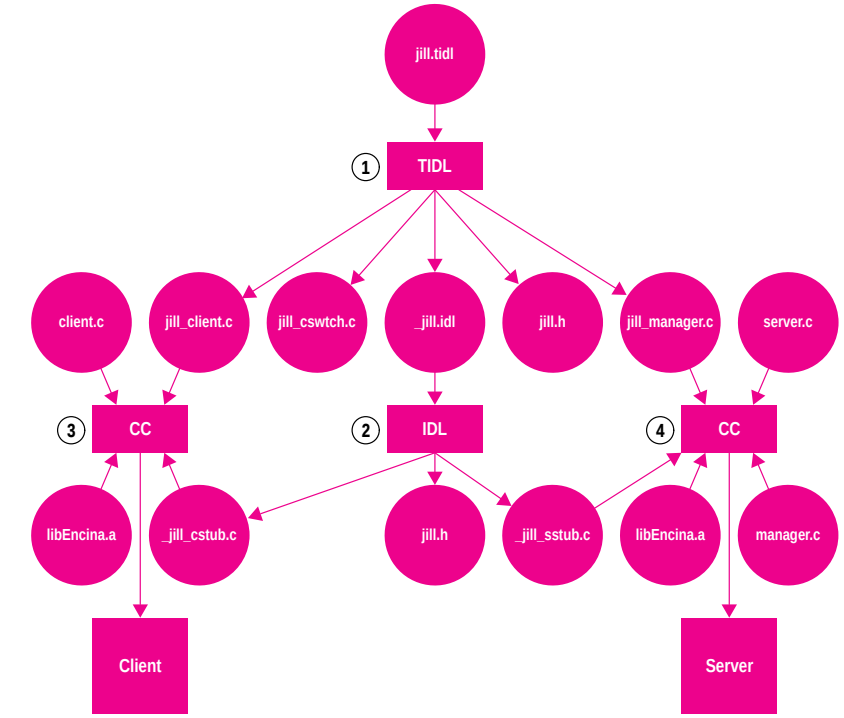
A transaction can make transactional RPC calls to multiple servers, and a server can in turn make a transactional RPC call to another server.

The transactional RPC service extends the DCE RPC model. The interface definition for the service executed on behalf of a transaction is defined in a TIDL (Transactional Interface Definition Language) file, which is similar to a DCE IDL file.* This file must be preprocessed with a TIDL compiler (similar to an IDL compiler). The TIDL preprocessor generates client stubs, server stubs, a header file, and an IDL file. The DCE IDL preprocessor is run on the IDL file to generate additional stubs and header files. The client and the server executables are generated by compiling and linking the various stub sources and libraries. This process is illustrated in Fig. 5.

Transactional RPC also supports nontransactional RPCs (a nontransactional RPC call can be made by calling the transactional RPC service). The TIDL file for the service interface must specify that the service is nontransactional.

Log. This component of Encina/9000 provides logging capabilities. It provides write-ahead logging (see glossary) for storing log records that correspond to updates to recoverable data and log records that correspond to transaction outcomes. The log records are used by the transaction manager to undo the effects of transactions that have aborted and to ensure that the committed transactions are durable.

* See the article on page 55 for more about IDL files.



1 Preprocess transaction through the TIDL compiler.
2 Run the IDL file created in 1 through the IDL compiler.
3 Create the client program.
4 Create the server program.

Fig. 5. The steps involved in turning a transaction into client and server executables.

For earlier versions of Encina/9000, the log service was implemented to provide a log server that could be used by many different clients to store log records. The latest version of Encina/9000 supports the log service as a library which is linked into the client code.

The log service supports archiving of log data for crash and media recovery. It also supports mirroring of data.

Lock. This component provides two-phase locking (see glossary) facilities to ensure the isolation and consistency properties of transactions. Applications can request locks on resources before accessing them, and the lock manager ensures that the lock request on behalf of a transaction will not be granted if another transaction holds a conflicting lock on that resource. Locks are released automatically when the transaction completes, and the application may also request early release of locks when it is safe to do so. The lock service also supports locking for nested transactions.

The locking service implements logical locking in which the programmer defines lock names and associates the lock names with physical resources. When a programmer wants to lock a physical resource, the logical lock name associated with that resource is specified in the call to the locking service.

In addition to supporting the conventional read/write locks, Encina/9000 also supports intention locks and instant duration locks. Intention locks are used to declare an intent to subsequently lock a resource. The use of intention locks can reduce the potential for deadlock among concurrent transactions. Instant duration locks are locks that are granted but not held and can be used to implement complex locking algorithms.

The lock service also provides the ability to determine if a transaction is deadlocked or not.

Volume. This component maintains the data storage in terms of logical data volumes. It provides the ability to manage very large files and view multiple physical disks as a virtual file. It also supports the ability to mirror a data volume transparently to the client. The volume component sometimes sacrifices speed for increased reliability and may be inappropriate for certain applications. This component is currently not used by the log component.

TM-XA. The Encina/9000 TM-XA component implements the X/Open XA interface. The XA interface is a bidirectional interface between a transaction manager and a resource manager such as a database. The XA interface provides a standard way for transaction managers to connect to databases.

The use of TM-XA with the Encina/9000 monitor is recommended. In this case the server registers each resource manager with a call providing the name of the resource manager and an associated switch structure,[4] which gives the Encina/9000 TM-XA component information about the resource manager. In addition, the server must also be declared as a recoverable server. TM-XA allows Encina/9000 to hook up with standard database products such as Oracle, Informix, and so on.

Glossary

The following are brief definitions of some of the transaction-related terminology used in the accompanying article.

Transactions and ACID

A transaction is the logical grouping of a user function performed as a unit so that it maintains its ACID (atomicity, consistency, isolation, and durability) properties. Transactions allow users to execute their programs and modify shared resources like databases in the presence of simultaneous access and updates by multiple users and in the presence of various kinds of failures.

Atomicity of a transaction means that either all the actions specified within the transaction will be performed or none of them will be performed. This ensures that a transaction is not partially applied, which is desirable since a partial application of the user transaction could leave the database in an inconsistent state. Consistency means that the database consistency is preserved in the presence of concurrency among multiple users. Isolation means that while the transaction is executing, its effects will not be visible to other concurrently running transactions. Durability means that once a transaction has been successfully completed the effects of that transaction are made permanent and survive failures.

Commit, Abort, and Prepare

A successful completion of a transaction is called a *commit* of the transaction. Before a transaction commits, it can be aborted either by the user or by the transaction processing system.

A user organizes a set of actions in a transaction with the intent that all of these actions should happen or none of these actions should happen. An example of such a transaction is a transfer of money between a person's savings account and checking account. This transaction consists of the actions of changing the savings account balance and the checking account balance. If the transaction is successful then both these account balances should be changed, one debited by amount X, and the other credited by amount X. Any other outcome would be in error. When the user submits this transaction and all operations in the transaction are successfully carried out, the transaction is said to have committed.

It may not always be possible to commit a transaction. For example, the machine that maintains the checking account balance may be down, or the user may have supplied an incorrect personal identification number (PIN) or decided to cancel the request after submitting it. If the transaction cannot be performed, then it is said to have been aborted or simply to have aborted. If a transaction is aborted (aborts), then none of the actions of the transaction are performed (or if they had been performed they are undone).

Transaction processing systems use mechanisms like two-phase locking to ensure the isolation properties, and *two-phase commit* protocols to ensure that all the participants within a transaction can be atomically committed or aborted.[1]

A two-phase commit protocol is typically used to commit a transaction in which multiple participants are performing actions requested by the transaction. One of these participants is called the coordinator. In the first phase of the two-phase commit protocol, the coordinator sends a *prepare* message to all the participants, asking them to prepare the transaction and send a message back indicating whether or not they can prepare the transaction. If all the participants respond that they can prepare the transaction, the coordinator writes a record in its log indicating that the transaction is committed and instructs all the participants to commit the transaction (this is the second phase of the protocol). If any participant responds back to the coordinator that it is unable to prepare the transaction, the coordinator notes in its log that the transaction is aborted and instructs all participants to abort the transaction.

When a participant receives a commit message from the coordinator in the second phase, it must ensure that all the actions of the transaction are durably stored on disk. When a participant receives an abort message from the coordinator in the second phase, it must ensure that all the actions of the transaction are undone. Therefore, in the first phase of the two-phase commit protocol, the participant must ensure that data is stored reliably in logs that enable it subsequently to undo the effects of a transaction or make permanent the effects of a transaction.

Nested Transaction

In the nested transaction model, a transaction (also called a top-level transaction) can be decomposed into a tree-like hierarchy. For example, a debit-credit transaction can be decomposed into two subtransactions, one for debit and one for credit. The debit or credit subtransactions could be further decomposed into smaller subtransactions. A subtransaction maintains the durability and consistency properties of its parents. The difference is that a subtransaction can be aborted and reexecuted without aborting its parent. In the debit-credit example, the debit subtransaction can be aborted and reexecuted without having to abort the entire transaction. In this case the top-level transaction verifies that each subtransaction can be committed at the time of transaction commit.

Two-Phase Locking

Two-phase locking means that a transaction will have two phases. In the first phase the transaction can only acquire and not release locks, and in the second phase it can only release and not acquire locks.

Write-Ahead Logging

For data recovery, transaction processing systems use a log to log data that is being modified. In write-ahead logging, data is copied to the log before it is overwritten. This ensures that if the transaction is aborted, the data can be restored to its original state.

Reference

1. J. Gray and A. Reuters, *Transaction Processing Concepts and Techniques*, Morgan Kaufman, 1993.

TM-XA allows an Encina/9000 application to make calls to one or more resource managers that support the XA interface. The Encina/9000 application starts a transaction, accesses a resource manager using its native SQL interface, and then commits the transaction. The TM-XA software coordinates the commit of the transaction among the various resource managers and other Encina/ 9000 components (like the structured file system or the recoverable queuing system) which might be accessed by the user transaction.

Recovery. This component drives the recovery protocols to recover from failures. It provides recoverable memory management. Recovery also provides the ability to perform:

- Abort recovery. This ensures that after a transaction is aborted, it is rolled back at all participating sites.
- Crash recovery. This provides a recovery after a system failure by rolling back all the transactions that had not been committed and rolling forward all the committed transactions.
- Media recovery. This is used to provide recovery when data written to the disk is destroyed.

The recovery component produces and uses the records written by the log service and ensures the consistency of transactional data. In the case of a transaction failure, the recovery component undoes the effects of a transaction. During recovery from a system failure, this component will examine the records in the log, appropriately commit or abort transactions for which it finds records in the log, and bring the data to a consistent state.

For media failures, the system administrator must provide archives that are used by the recovery component to restore the data to the state it was in when the archive was created.

There may be a loss of data in the case of media recovery. Encina/9000 also provides the ability to perform online backups to create the media archives necessary for recovery.

Tran-C

Encina/9000 provides extensions to the C programming language to make it easy to invoke the functionality provided by the Encina/9000 toolkit. Tran-C consists of library functions and macros that provide a simple programming paradigm so that the user does not have to access the toolkit module interfaces directly. The user can invoke high-level Tran-C constructs rather than the lower-level toolkit calls. The use of Tran-C versus toolkit calls is analogous to using a high-level language versus assembly language. Using the toolkit primitives directly is much more flexible, but the flexibility comes at the price of far greater complexity. In general, Tran-C is recommended for application programming.

The most important constructs provided by Tran-C are the transaction, onCommit, and onAbort clauses. These constructs provide a mechanism for the programmer to start a transaction and declare code to execute when the transaction commits or aborts. This is illustrated in Fig. 6. The application programmer is freed from the task of initializing all the underlying toolkit components and manually managing transaction identifiers, transactional locks, and other transactional metadata. All the code bracketed by the transaction clause is executed on behalf of the same transaction. When a transaction bounded by the transaction construct aborts (or commits), control in the program automatically transfers to the associated onAbort (or onCommit) clause.

Tran-C also supports nested transactions and multiple threads of control. The concurrent and cofor constructs can be used to create multiple concurrent threads within a transaction. The concurrent construct is used to enable an application to concurrently execute a predetermined number of threads, while the cofor construct enables the application to concurrently execute a variable number of threads. Both constructs provide the ability to create multiple threads which can be run either as subtransactions or as concurrent threads within the transaction. The subTran construct allows a created thread to be executed as a subtransaction within the parent transaction. The subThread construct allows a created thread to be executed as a separate thread within a nested transaction.

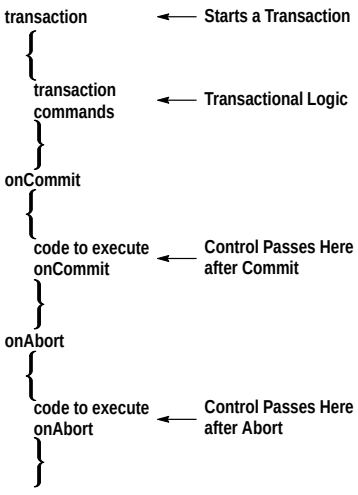


Fig. 6. A code fragment illustrating the use of the Tran-C constructs Transaction, onCommit, and onAbort.

Toolkit Example

Fig. 7 shows an example of the interactions between the components of the Encina/9000 toolkit. In this example a client makes a call to update data stored by a database and then commits the transaction. The following steps are associated with the circled numbers in Fig. 7.

1. The client starts a transaction by making a call to the transaction manager.
2. The client performs a transactional RPC by making a call to the transactional RPC component.
3. The transactional RPC component makes a call to the transaction manager to obtain transactional data for the transaction.
4. The transactional RPC component calls DCE RPC to transmit the user data and transactional data to the server.
5. DCE RPC (on the server) makes a call to the transactional RPC.
6. The transactional RPC component passes the transactional information to the transaction manager.

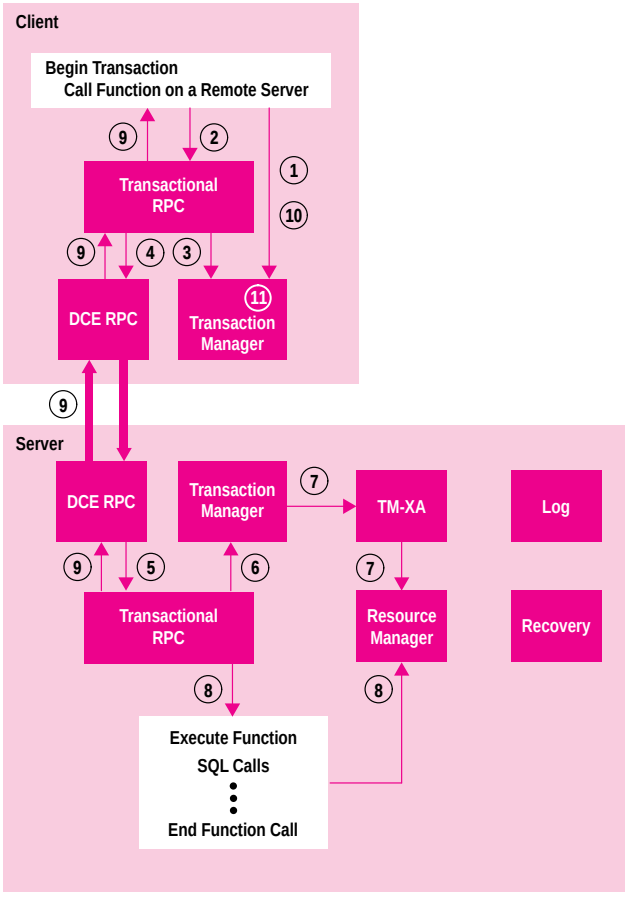


Fig. 7. An example of the interactions between components of the Encina/9000 toolkit.

7. The transaction manager uses the TM-XA interface to call the resource manager.
8. The transactional RPC component calls the user function that makes SQL calls to the resource manager. The resource manager performs the appropriate locking and updating of its data.
9. The user function on the server returns to that transactional RPC component which then returns to the client via DCE.
10. The client calls the transaction manager to commit the transaction.
11. The transaction manager uses a two-phase commit protocol to commit the transaction. It contacts all the transaction manager participants that have participated in the transaction. Each transaction manager uses the recovery and log components to log the prepare and commit decisions during various phases of the commit protocol for the transaction.

Peer-to-Peer Communications

Encina/9000 peer-to-peer communications, or PPC, provides transactional access to data stored on mainframes, and it performs a distributed two-phase commit of data stored on mainframes and HP 9000 servers. This allows mainframe applications to participate in an Encina/9000 transactional application, and conversely, an Encina/9000 application is able to participate in a mainframe transactional application. Encina/9000 PPC uses a two-phase commit sync protocol (sync level 2) to commit a transaction that accesses data on a mainframe and an HP 9000 server.

PPC services are implemented as a PPC executive and a PPC gateway product. These products can be purchased separately. The PPC executive is a library that runs in a DCE cell, and the PPC gateway is a server that acts as a gateway between DCE and SNA communications protocol. This gateway allows Encina/9000 applications to communicate with LU 6.2 applications.*

A typical PPC configuration involves an Encina/9000 PPC application running in a DCE cell and communicating with a PPC gateway server running in the same DCE cell. The PPC gateway server communicates with the mainframe using an SNA communications package. PPC provides the ability to write Encina/9000 applications that act as either the coordinator or the subordinate in a transaction between an Encina/9000 system and a mainframe host. Encina/9000 application programmers use the CPI-C API for coding the PPC component. The PPC gateway translates the CPI-C conversations from TCP/IP to LU6.2. This is illustrated in Fig. 8.

Structured File System

The structured file system is a record-oriented file system based on the X/Open ISAM standard. It provides an alternative to other commercial resource managers and the ability to support nested transactions that access data in the structured file system. It also provides full transactional integrity.

The records in the structured file system contain different fields that can be indexed by primary keys and secondary keys. The structured file system's field and record types are similar to those used by the recoverable queuing service (described below), allowing applications to have easy access to both systems. In addition, the structured file system supports a COBOL interface with the structured file system's external file handler.

* LU 6.2 applications are mainframe applications that are written to run on top of IBM's LU 6.2 protocol.

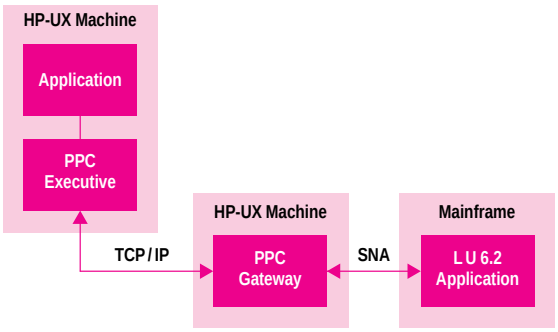


Fig. 8. A PPC configuration showing the PPC gateway translating TCP/IP protocol to SNA protocol.

Files in the structured file system are organized in one of the following three ways: entry-sequenced, relative, and B-tree clustered (see Fig. 9). Records in an entry-sequenced file are stored in the order in which they are written into the file. New records are always appended to the end of the file. A relative file is an array of fixed-length slots. Records can be inserted in the first free slot found from the beginning of the file, at the end of the file, or in a specified slot in the file. A B-tree clustered file is a tree-structured file in which records with adjacent index names are clustered together.

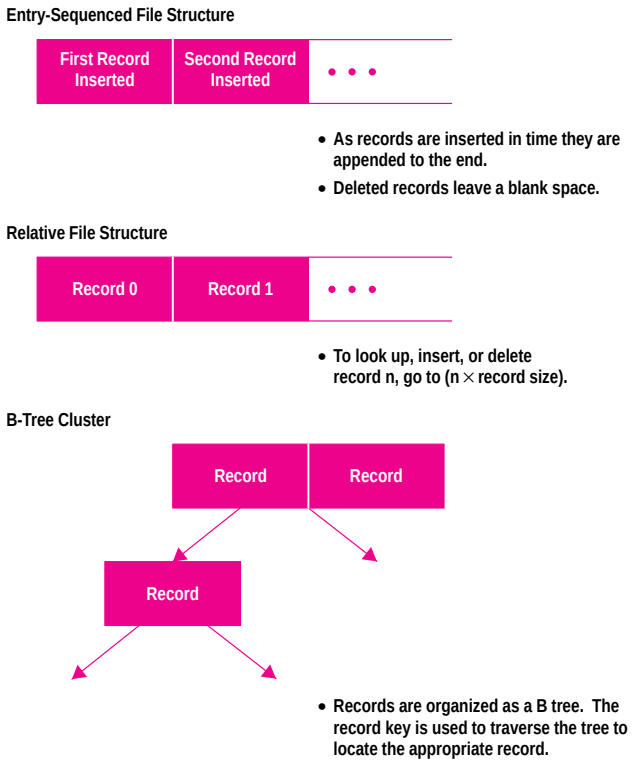


Fig. 9. File organizations supported in the structured file system.

The structured file system is simple and fast, but limited in flexibility when compared to relational databases. Relational databases provide powerful and complex access semantics with operations such as select, join, aggregate, and so on. The structured file system provides low-level access to records whose formats are user-defined and controlled.

Recoverable Queueing Service

Encina/9000 provides a recoverable queueing service which is layered on top of the basic toolkit and server core components. This service provides applications with the ability to transactionally queue and dequeue data. Application developers can write applications that transactionally update data in a resource manager like a database and queue or dequeue data with the guarantee that either both operations will succeed or both operations will abort.

An example of a transactionally recoverable queue would be a banking application that sends a letter to a customer if the customer's balance goes below zero. The action to generate the letter can be queued and processed later at the end of the day. The recoverable queue ensures that this action will always be performed even in the event of system failures.

One advantage of the queueing model is that applications can offload some work to be done at a later time. This deferred mode of computing is in contrast with the RPC style of communication in which an application invokes a service to do the processing as soon as it can.

A queue is a linear data structure. Elements in the data structure are queued in a particular configurable order and the dequeue occurs on a FIFO (first in, first out) basis. An element of a recoverable queueing service queue is structured in a record-oriented format. Encina/9000 supports queues that may contain elements of different data types. An element key is a sequence of one or more fields of an element type that are used to retrieve an element.

Encina/9000 provides the ability to define one or more recoverable queueing service servers in an Encina/9000 cell. Each server can internally support multiple queue sets. A queue set is a collection of queues within a recoverable queueing service server. Applications can queue or dequeue to or from a particular queue set. Queues within a queue set can be assigned priority classes relative to each other. Also, service levels define how to distribute the dequeues so that the queues with lower priority are not starved.

The recoverable queueing service supports a weak FIFO locking behavior. For example, when two transactions concurrently dequeue from a queue, each obtains a lock on the first element that it can lock on the queue. It is possible for the transaction that obtained a lock on the second element in the queue to commit before the transaction that obtained a lock on the first element in the queue. Another consequence of the weak FIFO locking policy may be that a transaction that consecutively queues multiple elements may not be able to place all these elements in that queue in an uninterrupted sequence.

The recoverable queueing service uses the DCE security mechanisms to secure access to the queue. Administratively, ACLs (access control lists) can be set up to authorize users or groups to be able to perform queue operations like read from queue, queue to the queue, dequeue from the queue, delete a queue, and so on.

A recoverable queueing service queue can be scanned using element keys, cursors (for sequential access), or element identifiers.

Finally, the recoverable queueing service provides the ability to register callbacks with the service's server on callback quantities such as the number of elements, size in bytes, and work accumulation. For example, with this feature it is possible to write applications that can ask the recoverable queueing service server to inform them when ten elements have been queued.

Monitor

The Encina/9000 transaction processing monitor provides an infrastructure for application development, run-time support, and administration. It supports the development of a three-tiered architecture in which multiple clients can access data stored in multiple resource managers.

Like DCE, the Encina/9000 monitor also has the concept of a cell. For the Encina/9000 monitor the cell is called an *Encina cell*. The Encina cell is a subset of the DCE cell, and multiple Encina cells can be defined within a DCE cell. (DCE cells are described in the article on page 6.) An Encina cell may consist of multiple nodes. A node is either a public node or a secure node. A secure node is a node on which the Encina/9000 servers can be securely run. Public nodes are nodes where only clients are run. Servers are not configured to run on public nodes. An example of an Encina cell is shown in Fig. 10. Like DCE, an Encina cell has a cell administrator who is responsible for performing administrative tasks.

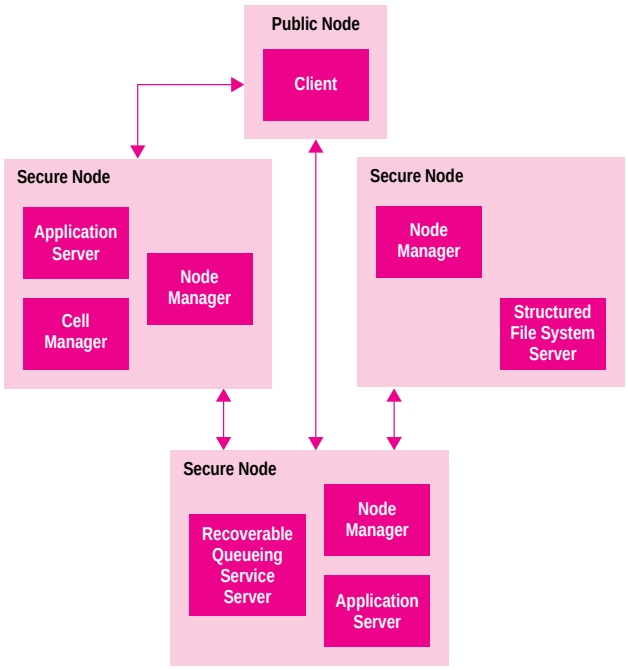


Fig. 10. The components in an Encina/9000 cell.

The encina cell contains the following server processes:

Cell Manager. There is one cell manager process in an Encina cell. The cell manager maintains the data needed to configure and administer the Encina cell. This data is stored in a data repository managed by the structured file system. The data describes how the application servers are configured to run on the secure nodes and includes the authorization information for those servers. The cell manager also monitors the state of the node managers and keeps statistics on the use of the servers by the clients.

Node Manager. There is one node manager process in each secure node in an Encina cell. The node manager monitors the application servers that are running in that node. If an application server fails, the failure is detected by the node manager which then restarts the application server.

Application Server. The server part of a user application is an application server. Typically, application servers accept calls from an Encina cell's clients and then process the user requests by accessing one or more resource managers. Application servers may be recoverable or ephemeral. A recoverable application server is one that uses the underlying Encina/9000 facilities to provide the ACID (atomicity, consistency, isolation, and durability) transactional properties. When a recoverable server fails, it performs recovery on restart which guarantees the consistency of the data. An ephemeral server does not provide the ACID properties to the data it accesses. An application server consists of a scheduling daemon process (called mond) and one or more processes (called PAs) that accept client requests. PAs are multithreaded processes. The mond coordinates the clients' requests for servers and assigns a PA to a requesting client. In this respect the role of the mond is similar to that of the RPC daemon rpcd in DCE. The mond also monitors the PAs and automatically restarts a PA in the event that the PA dies. An example of an application running in an Encina/9000 cell is shown in Fig. 11.

In the Encina/9000 monitor environment, a client can make a server request using explicit binding or transparent binding. With transparent binding the client simply makes a call to the server and the monitor environment is responsible for routing the client request to an appropriate server. With explicit binding, a client explicitly binds to a particular server. The Encina/9000 monitor provides a call to request a list of all servers exporting a particular interface, a call to get a handle to a mond for one of those servers, and a call to get a handle to a PA that is under the control of a particular mond. When using explicit binding, a client can specify that the client block if the PA is busy or that it get back a status if the PA is busy. In addition, the client can request that the PA be reserved for that client by specifying a long-term reservation to the server.

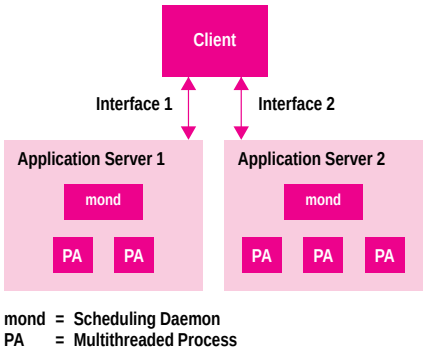


Fig. 11. An example of an encina cell's application servers.

In general it is easier to code the client to use transparent binding. This also has the advantage that the monitor code can perform load balancing of client requests among the available PAs. The monitor software uses a probabilistic algorithm to route client requests to the available PAs in a ratio predefined by a system administrator. With transparent binding the monitor software will use an existing binding if one exists, or it will create a binding to an appropriate server if no such binding exists. If all the available servers are busy, the client waits at the server for a free PA.

Although it is more complicated to write clients that use explicit binding, it does provide the user with the ability to select the PA on which the call is executed. There are certain situations in which explicit binding used in conjunction with long-term reservation of PAs is advantageous. For example, consider a client process servicing a large number of users. In this case it would be advantageous for that process to reserve a PA and then direct the various user requests to that PA. Having a direct connection to a PA reduces the time needed to connect to a PA on subsequent calls. Long-term reservation makes the PA unavailable to other clients, and it must be used with care. Administratively, a timeout interval can be specified so that if there are no client calls to the PA within that interval, the long-term reservation is canceled.

When an application server is initialized, it can specify one of the three scheduling policies: exclusive, concurrent shared, and shared. Shared scheduling is provided primarily for compatibility with previous releases, and its use is not recommended. The default policy is exclusive scheduling. With this scheduling only one client RPC can be executing within a given PA at any time, and the PA is scheduled exclusively for the entire duration of the client transaction. This has the advantage that the programmer does not have to be concerned about issues related to threading. This is required when the PA is accessing a database that is not thread safe (which is currently the case for most RDBMSs).

With concurrent shared scheduling, many clients can be executing within a PA at the same time, and the multithreaded PA assigns a different thread for each client request. If the PA accesses global or static variables, they must be protected by DCE synchronization facilities such as mutexes.* Concurrent shared scheduling should only be used when linking with thread-safe libraries. Concurrent shared scheduling provides the best performance with the lowest use of resources.

The monitor allows the creation and access of monitor-shared memory (HP 9000 virtual memory) which can be shared among the PAs within an application server. This allows a quick and easy way for the PAs to share transactional data. Monitor-shared memory is much cheaper than, say, using an external RDBMS, but care should be exercised when using the monitor-shared memory because it is the user's responsibility to perform the appropriate locking when accessing the shared memory. Since locks must be

* Mutexes, or mutual exclusion locks, are used to protect critical regions of code in DCE threads.

used, it also has the potential of introducing deadlocks. Transactional timeouts can be declared for aborting such transactions.

The monitor allows the use of the recoverable queueing service for queueing work items which are eventually processed by a monitor application server. Using the queued request facility, entries of the appropriate type are queued to the recoverable queueing service. A queued request facility daemon will then dequeue the request and forward the request to the appropriate PA.

The Encina/9000 monitor also provides a timer mechanism to allow servers to schedule a call to be issued at a later time. This functionality is provided transactionally so that the call made within the scope of a transaction is scheduled if the transaction commits. The call does not occur if the transaction aborts.

The Encina/9000 monitor provides support for application developers who wish to integrate their Encina/9000 client with forms-based user interface tools. Encina/9000 is integrated with JAM, a forms-based tool from JYACC Inc.

In summary, the Encina/9000 monitor provides the following benefits:

- Simplified programming for writing clients and servers
- Automatic detection of failures and restarts of monitor daemons and PAs
- Automatic load balancing between clients and servers
- Collection of statistics by the monitor for server use
- Simplified central place of administration for distributed clients and servers
- Support for highly concurrent access to relational databases.

Standards Supported by Encina/9000

Encina/9000 supports the following standards:

- X/Open:
  - XA
  - TX
  - TxRPC API
  - CPI-C
  - ISAM
- SAA:
  - CPIC
  - CPI/RR
- OSF DCE.

Encina/9000 interoperates with the following products:

- Oracle
- Informix
- Ingres
- Sybase
- Open CICS
- IBM mainframe CICS
- IBM mainframe IMS/DC.

The Encina/9000 toolkit has been used to support other transaction processing products and provide the base functionality to support other products like Open CICS and STDL each running on top of Encina/9000 on the HP-UX operating system.

Value-Added Features

HP Encina/9000 provides value-added features in the areas of system administration and high availability.

System Administration

An Encina/9000 system administrator must configure the Encina cell and define the administrative interfaces for the various servers in the system.

An Encina cell must be closely tied to a logical administrative unit of work, and the data accessed to do this work should be in the same cell. It is possible for applications to interoperate across Encina cells using explicit bindings. Therefore, the exact boundaries of an Encina cell must be defined by carefully analyzing the applications running in the system with careful consideration being given to security, number of users and machines, location of data, and the applications that access the data.

A system administrator must create the log space used by Encina/9000 and then bring up the following Encina/9000 components:

- Structured file system
- Cell manager
- Node manager
- Servers such as the recoverable queueing service and the PPC gateway if they are needed
- The required application servers.

Encina/9000 provides administration tools for the following components: log, structured file system, monitor, recoverable queueing service, PPC, and the rest of the toolkit. These tools provide the appropriate low-level commands for administering these components. Encina/9000 also provides a perl-based* tool called encsetup, which provides higher-level system administration facilities. The HP value-added system administration facilities are described later in this section. Finally, Encina/9000 also provides libraries for developing system administration products, which are very useful for customers developing these kinds of products.

Encina/9000 system administration is very closely tied to DCE system administration. The DCE cell must be configured before the Encina cell can be configured. Encina/9000 also makes use of the DCE directory service. The default Encina root cell directory is defined as /.:/encina (this default can be changed if needed). Encina/9000 components register their name under this directory. Within this directory there are directory entries for the recoverable queueing service, the structured file system, the Encina/9000 monitor, transactional RPC, and peer-to-peer communication (PPC). For example, each recoverable toolkit server registers an entry in the /.:/encina/trpc directory (trpc = transactional RPC), and each recoverable monitor server registers an entry in the /.:/encina/tpm/trpc directory (tpm = Encina/9000 monitor).

The use of the directory allows the Encina/9000 system administrator to restrict access to various resources. The

* Perl (Practical Extraction Report Language) is a UNIX programming language that is designed to handle system administrator functions.

system administrator can use DCE tools like acl_edit to grant a user, a group, or an organization permission to access a particular resource. Encina/9000 uses the DCE authentication and authorization mechanisms to maintain security. An Encina/9000 server can specify the level of authorization a user of the server must have to access that server. A client wishing to access a secure server must be authenticated with DCE and when the client calls the server, the server uses the DCE security mechanisms to verify whether it should allow access to the user. DCE access control and security are discussed in the articles of pages 49 and 41 respectively.

HP provides a DCAM layer for Encina/9000. DCAM stands for distributed computing application management. DCAM is an architecture and methodology for providing uniform system management for products that enable distributed computing such as DCE, Encina/9000, and CICS. An advantage of DCAM is that it provides a consistent look and feel for all of these products to the user and aids in the overall ease of use of these products. It provides a graphical user interface as well as a DCAM shell. DCAM provides a set of action verbs that can be modified by options and operate on objects.

Fig. 12 shows the relationship between the DCAM CLI (command-line interface) layer, the DCAM shell, and SAM (system administration manager).

SAM is a menu-driven interface used to manage an Encina/9000 system. The DCAM shell is a command-line interface which can be used to type in administrative commands. SAM and the DCAM shell are layered on top of the DCAM CLI scripts which convert the DCAM commands to native Encina/9000 administration commands.

The common look and feel provided by DCAM enables a system administrator to manage the different distributed systems and applications based on DCE with a consistent and user-friendly interface. DCAM does this by providing consistent use of vocabulary to represent actions. The consistent use of syntax and semantics is important because of the different subsystems that DCAM is built upon. The consistency provided by DCAM improves user efficiency and lowers error rates.

DCAM provides a natural way for system administrators to express the actions that they want. For example, to create a structured file system server, a system administrator would type the command: create sfsserver. This command is converted by DCAM to the underlying Encina/9000 low-level commands needed to create the server.

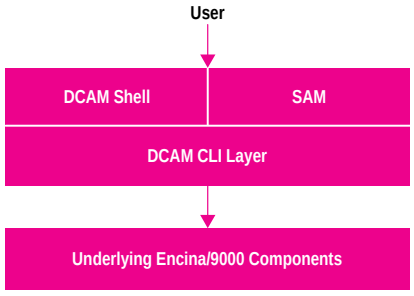


Fig. 12. System administration tools with DCAM.

The SAM interface of DCAM is more useful for people who are familiar with SAM or are getting acquainted with Encina/9000. The DCAM shell is generally used for efficiency by experienced Encina/9000 system administrators. In addition, the DCAM shell is also used for writing and customizing system administration scripts.

DCAM is object-oriented. Objects represent items that can be encapsulated and acted upon. Encina/9000 objects can be an Encina cell, a server, or a transaction. Objects have attributes. For example, a structured file system server has an associated attribute that describes the log volume associated with the server. Actions are verbs that act upon the objects. For example, the actions create, start, modify, and stop can be used to act upon an object. Actions have object independent semantics in that they have similar semantics regardless of the type of object they are working on. For example, the verb create can be used to create an Encina cell, a structured file system server, an application server, and so on. Actions have options. An action can be specified with the default options, or the administrator can specify task-specific options with the action.

A task defines a pairing of an action with an object. A task consists of one action, one object, zero or more options, and one or more attributes. For example, start cell -Name name, which tells DCAM to start up the named cell along with other optional parameters, is a task that can be specified with the DCAM shell. If the parameters are not specified, the DCAM shell will prompt for the parameters. In SAM, the parameters are displayed as fields in the SAM panel and can be entered. If the required parameters are not entered, an error is displayed.

Another useful feature of DCAM is the help facility, which can be used by the system administrator to interactively obtain help on a topic. This is also useful for someone who is learning Encina/9000 administration since it lists the various alternatives and options to a command and provides an easy way for administrators to get a feel for the various commands and options.

To many users the real value of DCAM is the added capabilities it has that go beyond what native Encina/9000 administration supports. This includes high-level server configuration tasks which are much easier, complete support for transparent remote configuration from anywhere in the DCE cell, autorestart of toolkit servers like the structured file system and the recoverable queueing service, and support for ServiceGuard/UX's failover* feature.

High-Availability Features

Many customers have a strict requirement for data to be available at all times. Data replication with Encina/9000 can be provided by the use of data mirroring with mirrored disks. In addition, to provide data availability in the case of machine failures, Encina/9000 can be integrated with the Switchover/UX and the ServiceGuard/UX products (described below). These products allow node failures to be handled, and they provide a set of scripts that facilitate the administration of a highly available system.

* Failover refers to the process that occurs when a standby node takes over from a failed node.

In a distributed system there can be many causes of failures, and failures of disks, networks, and machines can all impact availability. Since the system is composed of several nodes connected with network links, there are more points of failure that could impact availability. Network failures are not described here, but users who need highly available Encina/9000 applications should try to avoid single points of network failure.

Many techniques exist for dealing with disk failures. The preferred method of dealing with disk failures is to use HP-UX mirrored disks with a logical volume manager. Other choices are to use RAID[5] or to use Encina/9000 mirrored disks. The advantage of the HP-UX mirrored disk technique is that it is a general-purpose solution with applicability to all kinds of data like the structured file system and DBMSs. If the database can handle the logical volume manager configured for no consistency then it should be used for database data. Mirror write consistency, or mirror consistency, should be used for Encina/9000 data or for database data that can handle consistency mirroring. RAID can be used as a relatively inexpensive solution to handle disk failures, but it has many single points of failure in the disk I/O path and is not good for the short random write updates that are typically found in transaction systems. Encina/9000 mirroring has the disadvantage that it is not integrated in the HP-UX operating system and can therefore only be used for Encina/ 9000 data and not for, say, DCE or DBMS. Its advantage is that it can automatically handle more failure conditions than HP-UX mirroring. Encina/9000 mirroring is slower than HP-UX mirroring, but it has a faster recovery time.

There are two primary solutions for node failures: Switchover/UX and ServiceGuard/UX. In Switchover/UX, a primary node and a standby node are configured with multihosted disks. The primary node runs in the normal case. The standby node is also connected to the disks and uses a heartbeat protocol to detect failure of the primary node. When the standby node detects that the primary node has failed, it assumes the primary node's identity by booting off the primary node's disks and using the primary node's IP address. The standby node then uses the primary node's disks to reboot and to restart the system processes and applications. This allows a fast restart after the primary node has crashed, resulting in a small downtime. The primary and standby nodes should be from the same hardware family.

With Switchover/UX the Encina/9000 processes are restarted when the standby node reboots. Using Encina/9000's transparent binding, clients are automatically reconnected to the servers. However, in this case client transactions will keep aborting until the failover is complete.

In ServiceGuard/UX, applications and data are encapsulated as packages that can be run on various nodes of a cluster. ServiceGuard/UX allows the user to define the packages, and each package has a prioritized list of nodes it can run on. ServiceGuard/UX ensures that a package only runs on one node at a time. A package is defined by a startup/shutdown script and can represent any application. The nodes running packages monitor each other's status and restart packages when they detect the failure of another node.

A package can be an Encina/9000 application server running under a single Encina/9000 node manager. The package can also include assorted toolkit servers like the structured file system, a recoverable queueing service, or an Encina/9000 monitor. Optionally, a package can have one or more IP addresses. If specified, a package's IP address is associated with the network interface on the machine currently executing the package. With ServiceGuard/UX a user can configure a simple failover scheme. The user can also define a single package that can execute on a primary or a backup node. This scheme is general and can be used for the Encina/9000 log, structured file system, recoverable queueing service, monitor, and DBMS and DCE core servers.

Encina/9000 servers can be integrated with ServiceGuard/UX. In this case the Encina/9000 servers should be configured in a ServiceGuard/UX cluster, and a package should be created for the servers. The package should contain run and halt scripts for the servers, which specify the actions to take when a package is started or terminated. The actions in a run script include adding the relocatable IP address to the network interface, mounting all logical volumes, and calling an Encina/9000 script to start all the Encina/9000 servers. The actions in a halt script include calling an Encina/9000 script to halt all the Encina/9000 servers, unmounting all the logical volumes, and removing the relocatable IP address.

ServiceGuard/UX offers a more flexible solution for high availability. It can be configured with a dedicated standby solution similar to Switchover/UX, or it can be configured in a more cluster-like configuration. It also has a faster recovery time since failover nodes do not need to reboot.

Encina/9000 also provides the ability to perform application-level replication of data. An alternative to application-level replication is the replication of data provided by databases. Database-level replication has the advantage of being transparent to the user, and it is relatively efficient. Application-level replication, on the other hand, is less dependent on specific DBMS platforms and can be used to provide replication across DBMS platforms. In addition, it is more flexible and can be performed in a synchronous or asynchronous manner. It may be important to perform asynchronous replication across a WAN to achieve a faster response time. The disadvantage of application-level replication is that the application developer must design and implement the replication scheme.

An example of replication using Encina/9000 is master/slave replication of data with deferred updates to the slave. In this scheme, the master copy of the database is maintained on a machine. The application updates the master database and stores a copy of the update in the recoverable queueing service. With this setup the application can transactionally update the master database and store a copy of the updates in the recoverable queueing service. At a later time the data is transactionally dequeued from the recoverable queueing service and applied to the slave database on another machine. The strength of this approach is that the two machines holding copies of the data do not have to be running at the same time, and the update can be deferred to a time when the load on the system is low. It also avoids having to do online a two-phase commit across the machines. However, there is the drawback that the replica is not consistent with the master, and the updated data would be unavailable while the master node is down.

Encina/9000-Based Architectures

Some of the common Encina/9000-based architectures include corporate centralized data architecture, region centralized data architecture, and branch data architecture. In each of these architectures we consider a corporation to be an entity that has a central data processing center located at its headquarters. The corporation's business is geographically spread over several regions, and each regional center has a data processing center. Each region also contains multiple branch offices, and the branch office has a number of users who are executing transactions. In the past, companies employed mainframes at the corporate headquarters, where all the data was maintained. This was expensive to maintain, and the response time got worse as the data on the mainframe increased.

In a corporate centralized data architecture, data is still maintained at the mainframe host. Connection to the mainframe is through gateway machines that run the Encina/9000 PPC executive. Depending on the availability requirements, the gateway machines could be implemented with the high-availability solutions mentioned earlier. One option would be not to have any machines at the branch offices or the region offices but rather to have PCs at these offices which talk directly to the mainframe. Alternately, the regional centers or the branches could have machines, and the regional machine could route a request from a branch to the corporate center. All the data is maintained at the corporate center and there is no local business data at either the regional offices or the branch offices. This architecture is shown in Fig. 13.

This architecture is useful if it is hard to replace the mainframe machines and data. Alternately, it may be possible to offload some of the data from the mainframe to machines running the HP-UX operating system at the corporate data center.

The regional centralized data architecture is similar to the corporate centralized data architecture, except that the data is partitioned across the regional data centers. The data could also be stored with the mainframe at the corporate data center. Clients can run at the branch machines or at the regional center. Optionally, there could be a database at either the corporate center or the regions that assists in routing a request to the appropriate regional center. This architecture is shown in Fig. 14.

In the regional centralized data architecture, Encina/9000 servers typically run on the regional machines. Clients can run at the branch or regional offices. Clients employ lookup mechanisms to locate the appropriate server and then make calls to the servers. An Encina/9000 PPC can be used to transactionally read or update data stored at the corporate center.

The regional centralized data architecture has the advantage of avoiding CPU bottleneck problems when a large number of transactions have to be processed on a single database. Since the databases are spread throughout the regions, they all can handle transactional access to the data allowing a higher volume of transactional traffic. In addition, if users frequently access data at the nearby regional center, network traffic will be localized.

In the branch data architecture, each branch maintains its local branch data. The data can also be aggregated and maintained at the corporate center, but users primarily access the data at a branch machine. Optionally, corporate or regional centers can maintain a cross-reference database to assist with routing a user request to the appropriate branch. This architecture is shown in Fig. 15.

The main advantage of this solution is the fast response time, since for most transactions data can be looked up locally and expensive two-phase commits over the WAN can be minimized. The drawback of this scheme is having to maintain a large number of databases and administering them.

Conclusion

The Encina/9000 product provides an application development environment for developing OLTP applications and the run-time support for running and administering the applications. Its strengths are the flexibility it provides for distributed OLTP applications compared to the traditional database products, and its strong integration with the HP DCE product. It provides an infrastructure for customers to write reliable

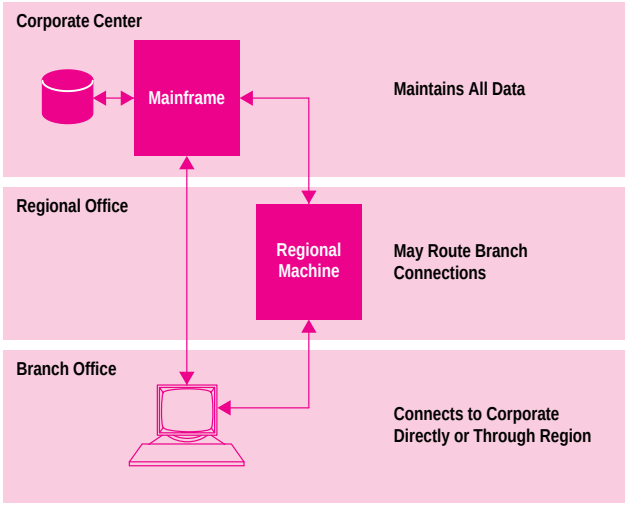


Fig. 13. A corporate centralized data architecture.

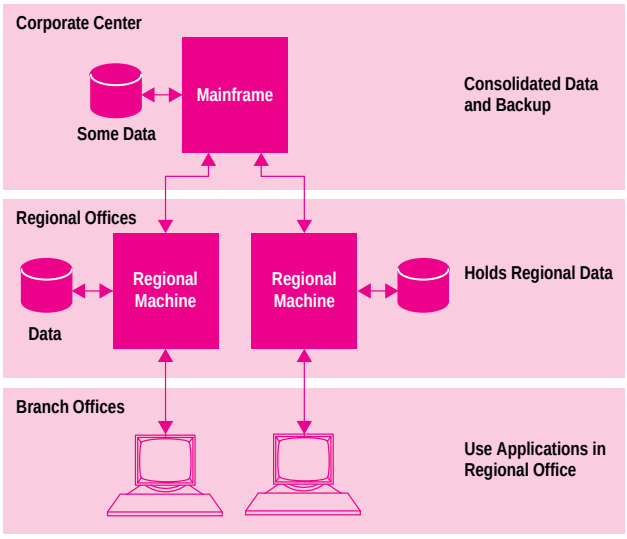


Fig. 14. A regional centralized data architecture.

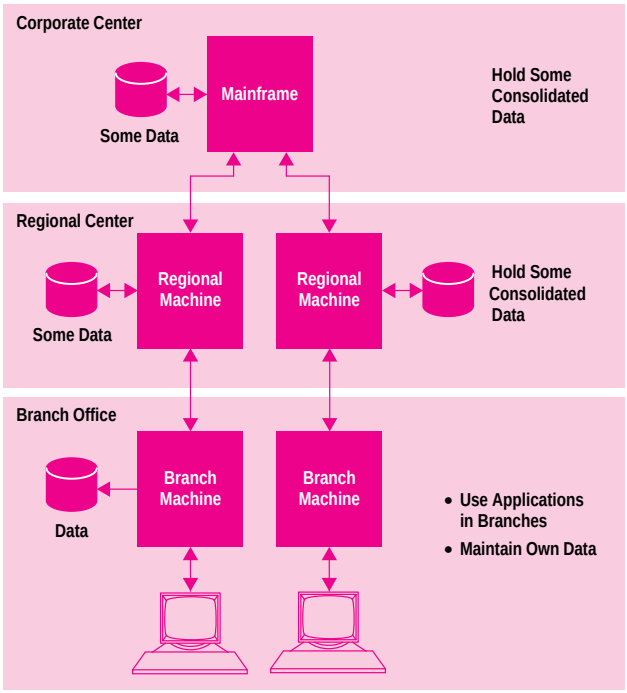


Fig. 15. Branch data architecture.

and secure applications for their mission-critical data. Additionally, the Encina/9000 product provides added value in the areas of system administration and fault tolerance.

Acknowledgments

I would like to thank Jay Kasi for his insightful discussions on several topics mentioned in this paper.

References:

1. *Encina/9000 Reference Manuals,* Part Number B 3789AA, Hewlett Packard Company, 1995.
2. *OSF DCE Application Development Guide*, Revision 1.03, Prentice Hall, 1993.
3. J.E.B. Moss, *Nested Transactions: An Approach to Reliable Distributed Computing,* MIT Press, 1985.
4. *CAE Specification Distributed Transaction Processing: The XA Specification,* X/Open, 1991.
5. M. Rusnack and T. Skeie, HP Disk Array: Mass Storage Fault Tolerance for PC Servers, *Hewlett-Packard Journal,* Vol. 46, no. 3, June 1995, pp. 71 to 81.